1. Write a blog on Difference between HTTP1.1 vs HTTP2

HTTP2 Vs. HTTP1 or HTTP1 Vs. HTTP2 – Let's Understand The Two Protocols

HTTP/1.1 has been around for more than a decade. With Google's SPDY leading the way in 2015, the IETF (Internet Engineering Task Force) gave us HTTP/2, which introduces several features to reduce page load times. Let's compare HTTP2 Vs. HTTP1.1 in detail.

HTTP/2 achieves faster webpage loading without performance optimizations that require extensive human efforts in terms of development. It significantly reduces the complexities that had crept into HTTP/1.1 and gives us a robust protocol which, though not without its flaws, will perhaps stand the test of time. Before making this leap forward, let's trace our steps back to when the internet was in its infancy to understand how the different versions evolved into the current form.

The Beginnings of HTTP & The Internet

Our story begins in 1969, with a program called Advanced Research Projects Agency Network (ARPANET). ARPANET used packet switching and allowed multiple computers to communicate with each other on a single network. However, this was just a by-product. The original intention behind ARPANET was to design a time-sharing system that allowed research institutes to share their computer resources for effective utilization of processing power.

Before then, sometime during the 19th century, the seeds for the existence of the internet as we know it today had already been sown with the invention of electricity and the telegraph. With Morse sending the first telegraphic message in 1844 and the first cable being laid across Atlantic, the telegraph network infrastructure had spread its roots through continents and across oceans. In years to come, this would become the very foundation on which the internet was built. In 1973, Kahn and Cerf designed the TCP/IP protocol suite which was adopted by ARPANET a decade later, and from this point on, we witness the development of an interconnected network. The internet took a more recognizable form with the invention of the World Wide

Web (that used HTTP as its underlying protocol) by Tim Berners-Lee and the Commercial Internet eXchange (CIX) that allowed a free exchange of TCP/IP traffic between ISPs.

Evolution of HTTP

HTTP (Hypertext Transfer Protocol) is a set of rules that runs on top of the TCP/IP suite of protocols and defines how files are to be transferred between clients and servers on the world wide web.

HTTP/1.x vs HTTP/2: A Comparative Study

HTTP2 Vs. HTTP1 is not a debate at all. HTTP2 is much faster and more reliable than HTTP1. HTTP1 loads a single request for every TCP connection, while HTTP2 avoids network delay by using multiplexing.

HTTP is a network delay sensitive protocol in the sense that if there is less network delay, then the page loads faster. However, an impressive increase in network bandwidth only slightly improves page load time. This is key to understanding the differences in performance efficiencies between the different versions of HTTP. Back in the day when people used dial up modems web pages were simple and it was the actual data transfer between the server and the client that contributed towards the largest chunk of the page load time. Today the actual downloading of resources from server takes a negligible portion of the total page load time due to the tremendous increase in bandwidth availability. It is the time taken to establish the TCP connection and making requests that impacts performance. It was initially recommended to use only two connections per hostname but today most browsers use six connections per hostname. When we talk about http vs http2 in terms of performance it is important to note that a lot of performance optimizations adopted by HTTP/1.1 introduced complexities in terms of developmental efforts as well as network congestion that HTTP/2 attempts to address.
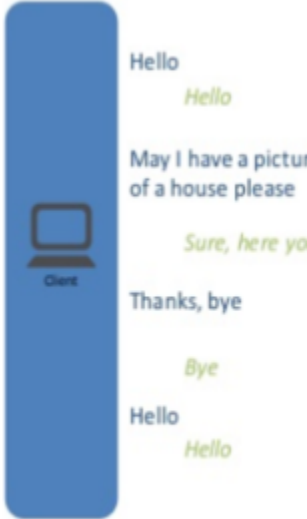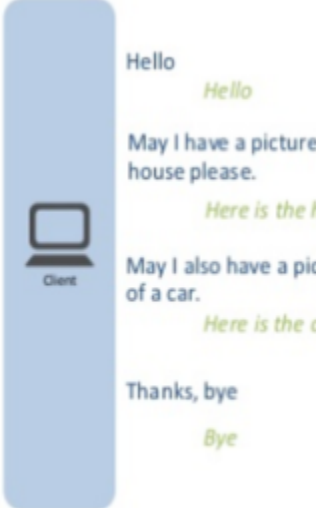
The table below points out the differentiating factors between http2 vs http1:

Header CompressionHeaders are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire.Header compression is included by default in HTTP/2 using HPACK.Performance OptimizationProvides support for caching to deliver pages faster.Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule.Removes the need for unnecessary optimization hacks.Protocol TypeText based protocol that is in the readable form.It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it.SecuritySSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication.Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since

almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

Header CompressionHeaders are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire.Header compression is included by default in HTTP/2 using HPACK.Performance OptimizationProvides support for caching to deliver pages faster.Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule.Removes the need for unnecessary optimization hacks.Protocol TypeText based protocol that is in the readable form.It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it.SecuritySSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication.Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

Differe ntiator	HTTP/1.0	HTTP/1.1	HTTP/2
Year	1991	1997	2015

Key Features	For every TCP connection there is only one request and one response. Hello *Hello* May I have a pictur of a house please *Sure, here you* Thanks, bye *Bye* Hello *Hello* **HTTP/1.0**	It supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution. Hello *Hello* May I have a picture house please. *Here is the h* May I also have a pic of a car. *Here is the c* Thanks, bye *Bye* **HTTP/1.1**	Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested. Hello *Hello* May I have a picture house please. And a car And a dog *Here is the hous* *Here is the car* And a cat *Here is the dog* *Here is the cat* Thanks, bye *Bye* **HTTP/2**
Status Code	Can define 16 status codes; the error prompt is not specific enough.	Introduces a warning header field to carry additional information about the status of a message. Can define 24 status codes, error reporting is quicker and more efficient.	Underlying semantics of HTTP such as headers, status codes remains the same.

Authentication Mechanism	Uses basic authentication scheme which is unsafe since username and passwords are transmitted in clear text or base64 encoded.	It is relatively secure since it uses digest authentication, NTLM authentication.	Security concerns from previous versions will continue to be seen in HTTP/2. However, it is better equipped to deal with them due to new TLS features like connection error of type Inadequate_Security.
Caching	Provides support for caching via the If-Modified-Since header.	Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags.	HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream.
Web Traffic	HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharding (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion.		HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays.

Header CompressionHeaders are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire.Header compression is included by default in HTTP/2 using HPACK.Performance Optimization Provides support for caching to deliver pages faster. Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule.Removes the need for unnecessary optimization hacks. Protocol Type Text based protocol that is in the readable form.It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. SecuritySSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication.Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

2.Write a blog about objects and its internal representation in Javascript

Objects And Its Internal Representation In JavaScript

Bablu Verma

Follow

Nov 5, 2020 · 4 min read

Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like `updateAddress`, `updateNam`, etc.

Objects and properties

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

`objectName.propertyName`

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let's

create an object named `myCar` and give it properties named `make`, `model`, and `year` as follows:

```
var myCar = new Object();
```

```
myCar.make = 'Ford';
```

```
myCar.model = 'Mustang';
```

```
myCar.year = 1969;
```

Unassigned properties of an object are `undefined` (and not `null`).

```
myCar.color; // undefined
```

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](#)). Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for

example, you could access the properties of the `myCar` object as follows:

```
myCar['make'] = 'Ford';
```

```
myCar['model'] = 'Mustang';
```

```
myCar['year'] = 1969;
```

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

```javascript
// four variables are created and assigned in a single go,

// separated by commas

var myObj = new Object(),

    str = 'myString',

    rand = Math.random(),

    obj = new Object();

myObj.type              = 'Dot syntax';

myObj['date created']   = 'String with space';

myObj[str]              = 'String value';

myObj[rand]             = 'Random Number';

myObj[obj]              = 'Object';
```

```
myObj['']                    = 'Even an empty
string';console.log(myObj);
```

You can also access properties by using a string value that is

stored in a variable:

```
var propertyName = 'make';
```

```
myCar[propertyName] = 'Ford';propertyName = 'model';
```

```
myCar[propertyName] = 'Mustang';
```

You can use the bracket notation with `for...in` to iterate over all

the enumerable properties of an object. To illustrate how this

works, the following function displays the properties of the object

when you pass the object and the object's name as arguments to

the function:

```
function showProps(obj, objName) {
```

```
var result = ``;
```

```
for (var i in obj) {
```

```
    // obj.hasOwnProperty() is used to filter out properties from
the object's prototype chain
```

```
  if (obj.hasOwnProperty(i)) {
```

```
    result += `${objName}.${i} = ${obj[i]}\n`;
```

```
  }
```

```
}
```

```
return result;
```

```
}
```

So, the function call `showProps(myCar, "myCar")` would return the following:

```
myCar.make = Ford
```

```
myCar.model = Mustang
```

```
myCar.year = 1969
```

Creating Objects In JavaScript :

Create JavaScript Object with Object Literal

One of easiest way to create a javascript object is object literal,
simply define the property and values inside curly braces as
shown below

```
let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};
```

Create JavaScript Object with Constructor

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

```
function Vehicle(name, maker) {

  this.name = name;

  this.maker = maker;

}

let car1 = new Vehicle('Fiesta', 'Ford');

let car2 = new Vehicle('Santa Fe', 'Hyundai')

console.log(car1.name);      //Output: Fiesta

console.log(car2.name);      //Output: Santa Fe
```

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example

var person = new Object();

person.firstName = "John";

person.lastName = "Doe";

person.age = 50;

person.eyeColor = "blue";

Using the `Object.create` method

Objects can also be created using the `Object.create()` method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

```
// Animal properties and method encapsulation

var Animal = {

 type: 'Invertebrates', // Default value of properties

 displayType: function() {  // Method which will display type of Animal

    console.log(this.type);

 }

};

// Create new animal type called animal1
```

```
var animal1 = Object.create(Animal);

animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes

var fish = Object.create(Animal);

fish.type = 'Fishes';

fish.displayType();

// Output:Fishes
```